University of Saskatchewan
Department of Computational Science

Cmpt 340
Final Examination

April 25, 1994

Time: 180 minutes (3 hours)
Total Marks: 90

Professor: A. J. Kusalik
Closed Book†

Name: _____

Student Number: _____

Directions:

Answer each of the following questions in the space provided in this exam booklet. If you must continue an answer (e.g. in the extra space on the last page, or on the back side of a page), make sure you clearly indicate that you have done so and where to find the continuation.

Where a discourse is called for (an explanation, justification, etc.), be concise and precise. Write your answers legibly at all times; no marks can be given for responses which cannot be decrypted.

Marks for each question are given at the beginning of that question. There are a total of 90 marks on the exam. Allocate your time accordingly.

Good luck.

A.	_____/10	E.	_____/10	I.	_____/25
B.	_____/10	F.	_____/5	J.	_____/5
C.	_____/5	G.	_____/5	K.	_____/5
D.	_____/5	H.	_____/5		

Total: _____/90

† Closed book, except for one optional 8.5x11 inch quick reference sheet ("cheat sheet") of the student's own compilation.

A. ***(10 marks)***
Short answer questions.

1. Indicate which of the following statements is the correct reading of the C++ declaration

```
int *&p1;
```

(a) "`p1` is a reference to a pointer to an object of type `int`."

(b) "`p1` is a pointer to a reference to an object of type `int`."

(c) "`p1` is a pointer to an object of type `int`."

(d) "The pointer is dereferenced, so `p1` is of type `int`."

(e) None of the above.

2. The ability to write meta-interpreters in functional and logic-programming languages (and thereby modify the language's semantics, or define new, related languages) is an example of satisfying what programming language design criterion?

3. What are the two domains involved in specifying the denotational semantics for a language?

4. In two types of semantic specification method, the idea of state (e.g. state of some environment or machine) is used. What are these two types of semantics?

5. When writing a meta-circular interpreter one can reify ("capture") the operations of the language at various levels. In particular, the interpretation can be at a high-level or a low-level of granularity. However, there is a trade-off between granularity and another consideration. What is that other consideration? I.e. the trade-off is between granularity and what?

B. ***(10 marks)***
Give an example to show that the programming language design principles of protection and modifiability can be mutually conflicting requirements in/for a C++ program. Explain how the principles are, in fact, conflicting within your example.

C. ***(5 marks)***
Consider a language implementation by way of a pseudo-interpreter. It is not uncommon to have the compiler for such a language implemented in the language itself. The purported advantage to this scheme is that it aids in portability. How can this be? Explain.

D. ***(5 marks)***
Does Prolog have data types? Explain your answer.

E. ***(10 marks)***
Insertion sort is a sorting algorithm which accumulates a sorted list from its input list by repeatedly inserting the elements of the input list into the partial sorted list in their correct positions. For example, given the input list `[4,1,5,3]`, the algorithm proceeds as follows:

input list	(partial) sorted list	
`[4,1,5,3]`	`[]`	
`[1,5,3]`	`[4]`	
`[5,3]`	`[1,4]`	
`[3]`	`[1,4,5]`	
`[]`	`[1,3,4,5]`	result = `[1,3,4,5]`

Give below a Miranda function which implements insertion sort. If you require auxiliary functions, provide their definitions as well.

F. ***(5 marks)***

Write a higher-order Miranda function `compose` which, given two functions of the appropriate type as arguments, returns a function which is their composition. I.e. if the arguments of compose are functions `f` and `g`, then compose computes `f o g`.

Hint: of course, `g` will be a function of one argument.

What is the type expression for function `compose`?

G. ***(5 marks)***

Consider a C++ expression involving class objects and operators on them; for example,

```
cout << "There are "<< q.len() << " elements in queue " << q << endl;
```

(assuming the appropriate operator and member function definitions for all classes involved). There is a similarity with expressions of this form and the idea of Currying in functional languages. What is that similarity?

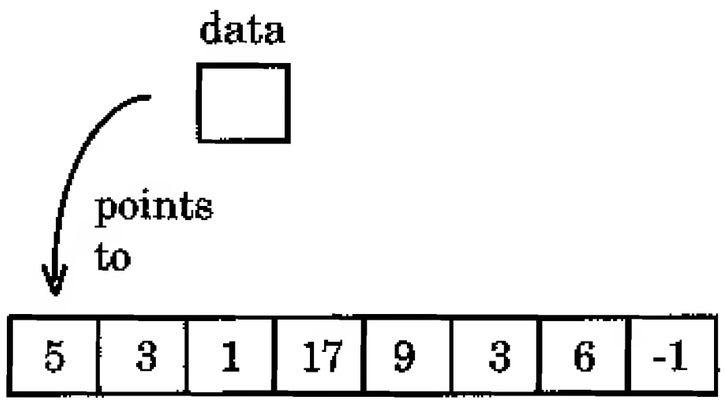
Hint to get you started: where would parenthesis be placed in the C++ expression above if the associativity was made explicit?

H. ***(5 marks)***

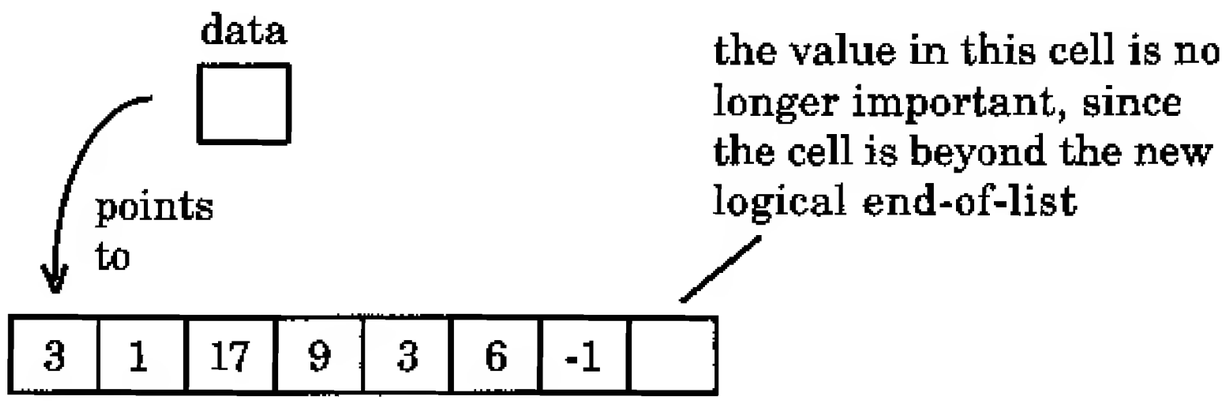
A segment of C++ code is required for "shifting up" the elements of an array of integers; i.e. the first element of the array is to be replaced by the second, the original second element by the third, the third by the fourth, etc. The logical end of the array is marked by an element with a negative value. It is assumed the array is pointed to by a variable `data`; i.e. we have previously a declaration:

```
int * data;
```

For example, if the situation prior to execution of the piece of code is:



then afterwards it should be:



The code written to perform this function (by a Cmpt 340 student) was:

```
for( int * tmp = data; (*tmp++ = *tmp) >= 0; );
```

where `tmp` had not be declared previously.

Is this piece of C++ code correct? I.e., will it implement the required function/functionality? If not, what is the problem, and how would you fix it?

I. ***(25 marks)***

The following is a list of program language constructs, features, and concepts available (present) in C++ (i.e. obtainable in C++ programs). Some of the terminology is specific to C++, and some is more generally used within the area of object-oriented programming languages.

Give an example of each concept, construct, or feature through a portion of C++ code. Make sure your example clearly illustrates the concept, construct, or feature. If necessary, indicate its occurrence within the example. Ensure that there is no ambiguity in what you are illustrating in each case.

You may use a different piece of code for each concept,construct, or feature, or you may use one code segment to illustrate several. If you have a problem thinking up example applications you could try the following ideas: complex numbers and arithmetic operations on them; geometric objects and operations on them; common data structures such as trees, lists, stacks, queues; simulations of real-world situations or environments; etc.

1. information hiding

2. instance variables

3. destructor

4. dynamic allocation

5. methods
(i.e. methods of a class of objects
or object)

6. default arguments

7. overloading

8. derived class

9. multiple inheritance

10. static binding

11. dynamic binding

12. abstract class

J. ***(5 marks)***
What is the weakest precondition for the following sequence of assignment statements and its post-condition?

```
a := 2*b+1;
b := a-3;
{b<0}
```

K. ***(5 marks)***
In the documentation on Prograph it is stated:

> "Prograph's dataflow language is data-driven. This means that the only prerequisite to an operation executing in Prograph is that its input data values be available at the input terminals of the operation when it begins to execute. If data is being generated elsewhere, the values must have already arrived along the datalinks."

What is the difference, then, between dataflow computation and lazy evaluation (as exemplified in a language like Miranda)? You may illustrate with an example.

And that's it! Have a good summer.

(extra space for answering previous questions or for rough work)